

# **Отчёт по лабораторной работе №11**

**Программирование в командном процессоре ОС UNIX. Ветвления и  
циклы**

Щербак Маргарита Романовна

2022

# 1 Цель работы:

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 1.1 Задание:

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
  - `-iinputfile` — прочитать данные из указанного файла;
  - `-ooutputfile` — вывести данные в указанный файл;
  - `-p` шаблон — указать шаблон для поиска;
  - `-C` — различать большие и малые буквы;
  - `-n` — выдавать номера строк.а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.
2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию о коде завершения в оболочку. Командный файл должен вызывать эту программу и выдавать сообщение о том, какое число было введено.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например `1.tmp`, `2.tmp`, `3.tmp`, `4.tmp` и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы

командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).

4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

## 2 Теоретическое введение:

### *Командные процессоры (оболочки)*

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.

В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: - оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; - C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; - оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; - BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный.

### 3 Выполнение лабораторной работы:

1. **Первое задание.** Для выполнения первого задания я создала файл prog1.sh, в котором буду писать скрипт, открыла текстовый редактор emacs. Также дала созданному файлу право доступа на выполнение (+x).

Считывать данные я буду из файла conf.txt, поэтому просмотрели его содержимое перед выполнением задания.

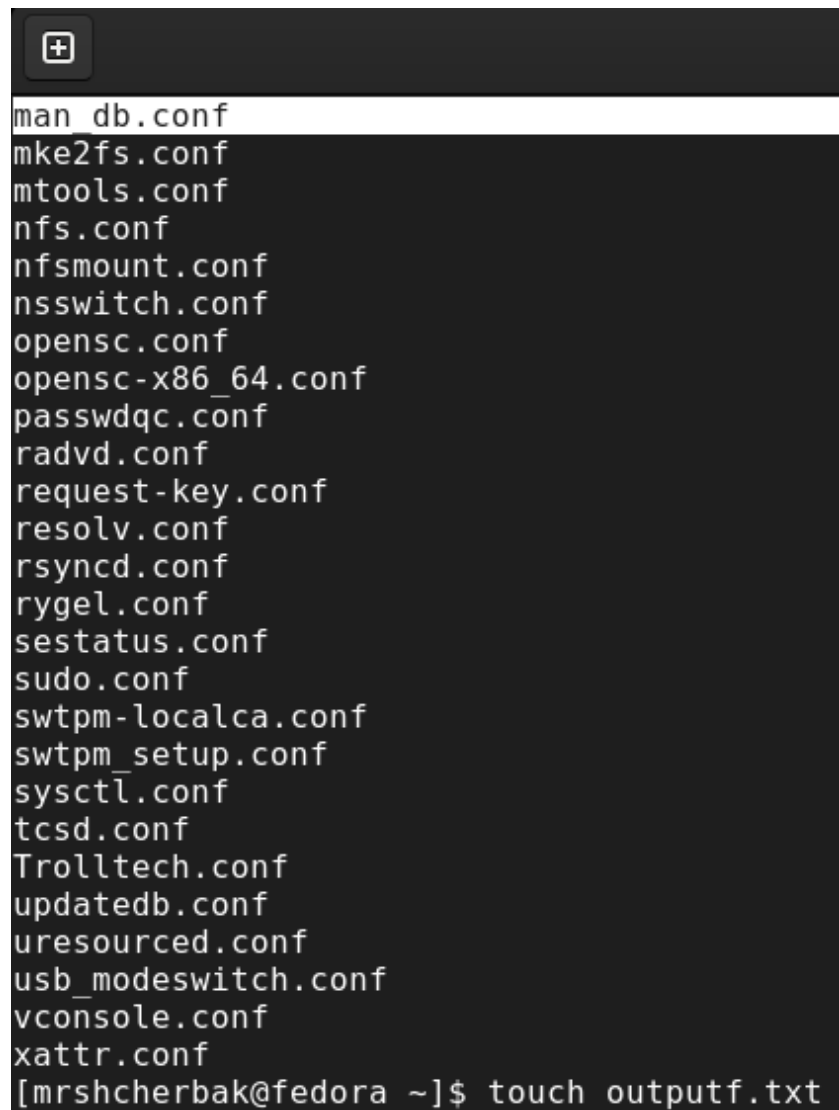
Создала файл outputf.txt, в который считывала содержимое файла conf.txt.

При выполнении задания взяла в качестве шаблона из файла conf.txt строку №23: man\_db.conf, “db” будет служить шаблоном для параметра -C.

(Рис. 3.1 - Рис. 3.5).

```
[mrshcherbak@fedora ~]$ touch prog1.sh
[mrshcherbak@fedora ~]$ emacs &
[1] 36970
[mrshcherbak@fedora ~]$ chmod +x prog1.sh
[mrshcherbak@fedora ~]$ cat conf.txt
anthy-unicode.conf
appstream.conf
asound.conf
brltty.conf
chrony.conf
dley-na-renderer-service.conf
dley-na-server-service.conf
dnsmasq.conf
dracut.conf
extlinux.conf
fprintd.conf
fuse.conf
host.conf
idmapd.conf
jwhois.conf
kdump.conf
krb5.conf
ld.so.conf
libaudit.conf
libuser.conf
locale.conf
logrotate.conf
man_db.conf
mke2fs.conf
mtools.conf
```

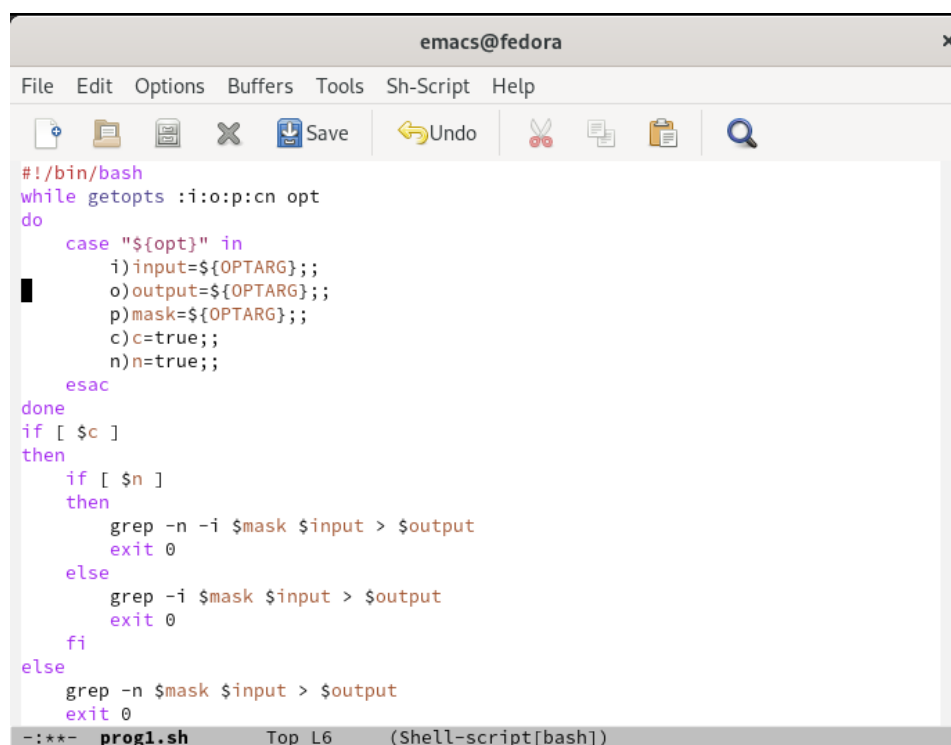
Рис. 3.1: Создание файла, в котором написан скрипт первого задания, предоставление права доступа на выполнение этому файлу и просмотр содержимого файла, который будем читать.



```
man_db.conf
mke2fs.conf
mtools.conf
nfs.conf
nfsmount.conf
nsswitch.conf
opensc.conf
opensc-x86_64.conf
passwdqc.conf
radvd.conf
request-key.conf
resolv.conf
rsyncd.conf
rygel.conf
sestatus.conf
sudo.conf
swtpm-localca.conf
swtpm_setup.conf
sysctl.conf
tcsd.conf
Trolltech.conf
updatedb.conf
uresourced.conf
usb_modeswitch.conf
vconsole.conf
xattr.conf
[mrshcherbak@fedora ~]$ touch outputf.txt
```

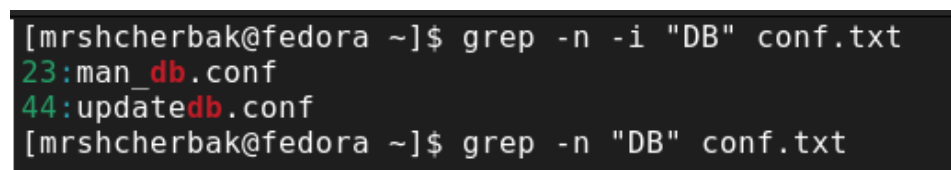
Рис. 3.2: Создание файла, в который будем выводить данные из указанного файла.





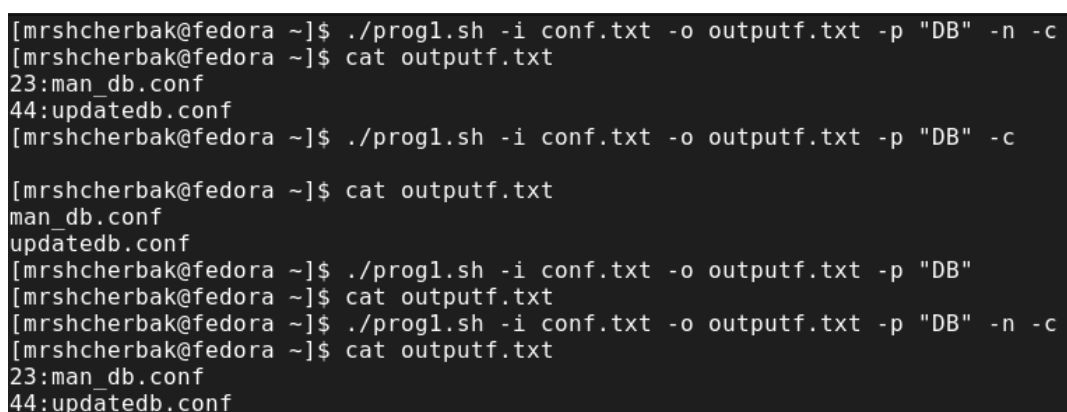
```
#!/bin/bash
while getopts :i:o:p:cn opt
do
    case "${opt}" in
        i) input=${OPTARG};;
        o) output=${OPTARG};;
        p) mask=${OPTARG};;
        c) c=true;;
        n) n=true;;
    esac
done
if [ $c ]
then
    if [ $n ]
    then
        grep -n -i $mask $input > $output
        exit 0
    else
        grep -i $mask $input > $output
        exit 0
    fi
else
    grep -n $mask $input > $output
    exit 0
fi
```

Рис. 3.3: Скрипт 1го задания



```
[mrshcherbak@fedora ~]$ grep -n -i "DB" conf.txt
23:man_db.conf
44:updatedb.conf
[mrshcherbak@fedora ~]$ grep -n "DB" conf.txt
```

Рис. 3.4: Выполнение



```
[mrshcherbak@fedora ~]$ ./prog1.sh -i conf.txt -o outputf.txt -p "DB" -n -c
[mrshcherbak@fedora ~]$ cat outputf.txt
23:man_db.conf
44:updatedb.conf
[mrshcherbak@fedora ~]$ ./prog1.sh -i conf.txt -o outputf.txt -p "DB" -c
[mrshcherbak@fedora ~]$ cat outputf.txt
man_db.conf
updatedb.conf
[mrshcherbak@fedora ~]$ ./prog1.sh -i conf.txt -o outputf.txt -p "DB"
[mrshcherbak@fedora ~]$ cat outputf.txt
[mrshcherbak@fedora ~]$ ./prog1.sh -i conf.txt -o outputf.txt -p "DB" -n -c
[mrshcherbak@fedora ~]$ cat outputf.txt
23:man_db.conf
44:updatedb.conf
```

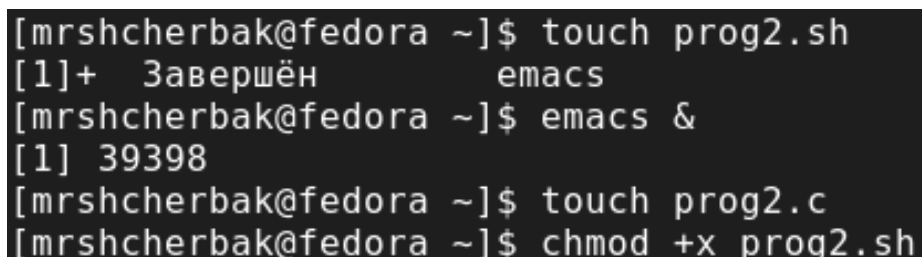
Рис. 3.5: Проверка работы скрипта с параметрами (ключами)

Делаем вывод, что скрипт работает корректно, используя команды `getopts` `grep`, я написала командный файл, который анализирует командную строку с ключами и ищет в указанном файле нужные строки, определяемые ключом `-p`.

2. **Второе задание.** Создала файл `prog2.sh`, в котором писала второй скрипт, и открыла его в редакторе `emacs`. Также создала файл `prog2.c`, в котором писала на языке Си программу, выводящую число и определяющую, является ли оно больше нуля, меньше нуля или равно нулю.

Предоставила право доступа на выполнение файлу `prog2.sh`.

(Рис. 3.6 - Рис. 3.9).



```
[mrshcherbak@fedora ~]$ touch prog2.sh
[1]+  Завершён          emacs
[mrshcherbak@fedora ~]$ emacs &
[1] 39398
[mrshcherbak@fedora ~]$ touch prog2.c
[mrshcherbak@fedora ~]$ chmod +x prog2.sh
```

Рис. 3.6: Создание файла, в котором написан скрипт второго задания, предоставление права доступа на выполнение этому файлу

```
mrshcherbak@fedora:~  
emacs@fedora  
File Edit Options Buffers Tools C Help  
[Icons]  
#include <stdio.h>  
#include <stdlib.h>  
int main(){  
    printf("Введите число: ");  
    int a;  
    scanf("%d", &a);  
    if (a<0) exit(0);  
    if (a>0) exit(1);  
    if (a==0) exit(2);  
    return 0;  
}  
U:--- prog2.c All L12 (C/*l Abbrev)
```

Рис. 3.7: Программа на языке Си

```
mrshcherbak@fedora:~  
emacs@fedora  
File Edit Options Buffers Tools Sh-Script Help  
[Icons]  
#!/bin/bash  
gcc prog2.c -o prog2  
./prog2  
code=$?  
case $code in  
    0) echo "Число меньше 0";;  
    1) echo "Число больше 0";;  
    2) echo "Число равно 0";;  
esac  
U:--- prog2.sh All L9 (Shell-script[sh])
```

Рис. 3.8: Скрипт 2го задания

```
[mrshcherbak@fedora ~]$ ./prog2.sh
Введите число: 16
Число больше 0
[mrshcherbak@fedora ~]$ ./prog2.sh
Введите число: 0
Число равно 0
[mrshcherbak@fedora ~]$ ./prog2.sh
Введите число: -69
Число меньше 0
[mrshcherbak@fedora ~]$
```

Рис. 3.9: Проверка выполнения скрипта

Таким образом, мы видим, что задание выполнено успешно.

3. **Третье задание.** Создала файл prog3.sh, в котором писала третий скрипт, и открыла его в редакторе emacs (C-x C-s). Предоставила право доступа на выполнение файлу prog3.sh. (Рис. 3.11).

Написала командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Этот же командный файл удаляет все созданные им файлы (Рис. 3.10).

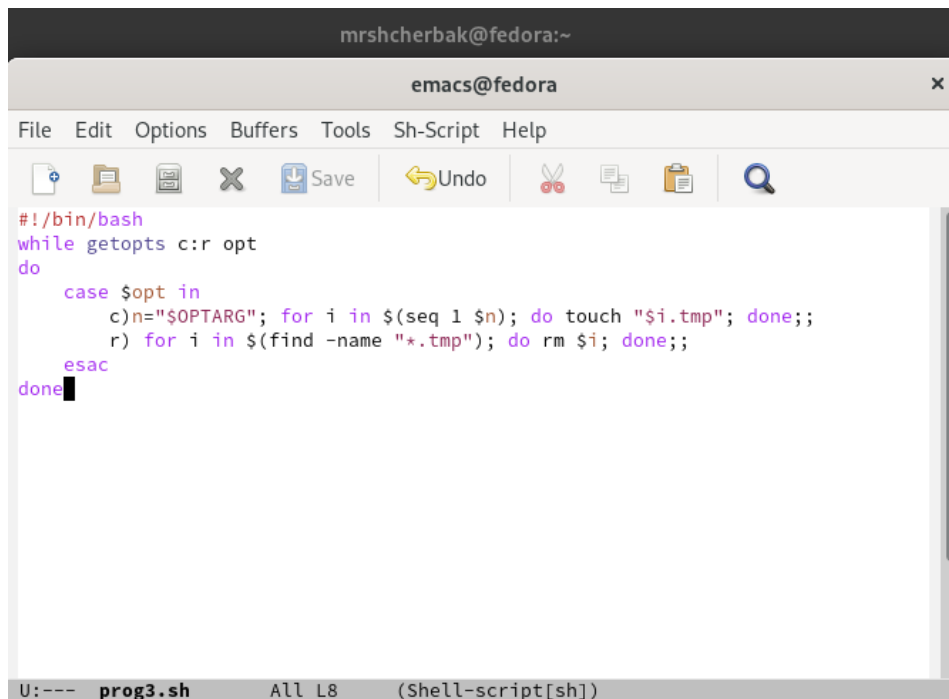


Рис. 3.10: Скрипт 3го задания



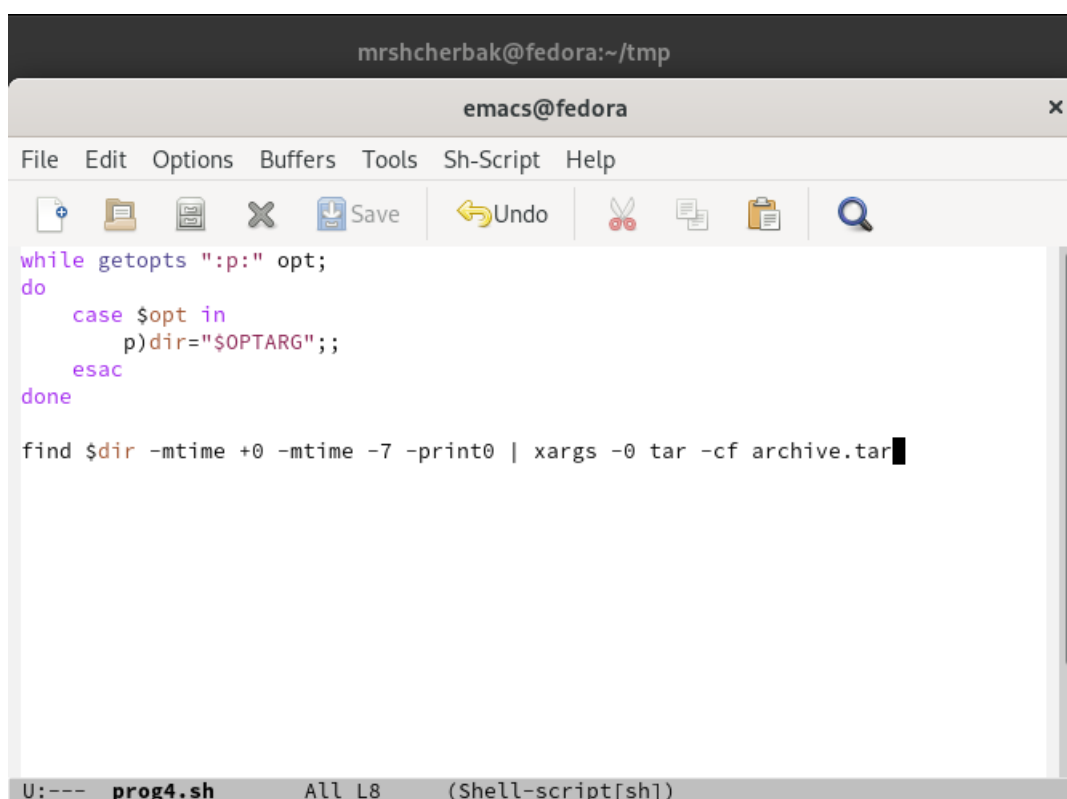
Рис. 3.11: Проверка выполнения скрипта

Скрипт работает корректно.

- Четвёртое задание.** Создала файл prog4.sh, в котором писала четвёртый скрипт, и открыла его в редакторе emacs (C-x C-s). Предоставила право доступа на выполнение файлу prog4.sh. (Рис. 3.13).

Написала командный файл, который с помощью команды tar запаковывает

в архив все файлы в указанной директории. Запаковываются те файлы, которые были изменены менее недели тому назад. (Рис. 3.12).

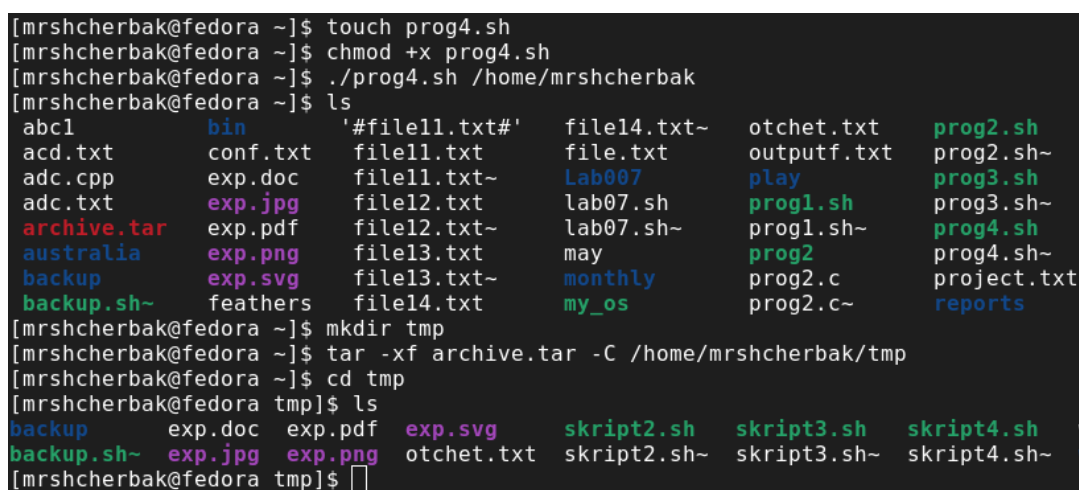


The screenshot shows an Emacs editor window titled 'emacs@fedora' with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Help) and a toolbar. The main text area contains a shell script. The status bar at the bottom indicates 'U:--- prog4.sh All L8 (Shell-script[sh])'.

```
while getopts ":p:" opt;
do
    case $opt in
        p)dir="$OPTARG";;
    esac
done

find $dir -mtime +0 -mtime -7 -print0 | xargs -0 tar -cf archive.tar
```

Рис. 3.12: Скрипт 4го задания



The screenshot shows a terminal window with the following commands and output:

```
[mrshcherbak@fedora ~]$ touch prog4.sh
[mrshcherbak@fedora ~]$ chmod +x prog4.sh
[mrshcherbak@fedora ~]$ ./prog4.sh /home/mrshcherbak
[mrshcherbak@fedora ~]$ ls
abcl          bin          '#file11.txt#'  file14.txt~   otchet.txt    prog2.sh
acd.txt       conf.txt     file11.txt      file.txt      outputf.txt   prog2.sh~
adc.cpp       exp.doc      file11.txt~     Lab007        play          prog3.sh
adc.txt       exp.jpg      file12.txt      lab07.sh      prog1.sh      prog3.sh~
archive.tar   exp.pdf      file12.txt~     lab07.sh~     prog1.sh~     prog4.sh
australia     exp.png      file13.txt      may           prog2         prog4.sh~
backup        exp.svg      file13.txt~     monthly       prog2.c       project.txt
backup.sh~    feathers    file14.txt      my_os        prog2.c~     reports
[mrshcherbak@fedora ~]$ mkdir tmp
[mrshcherbak@fedora ~]$ tar -xf archive.tar -C /home/mrshcherbak/tmp
[mrshcherbak@fedora ~]$ cd tmp
[mrshcherbak@fedora tmp]$ ls
backup        exp.doc      exp.pdf  exp.svg  skript2.sh  skript3.sh  skript4.sh
backup.sh~    exp.jpg      exp.png  otchet.txt  skript2.sh~  skript3.sh~  skript4.sh~
[mrshcherbak@fedora tmp]$
```

Рис. 3.13: Проверка работы скрипта

В папке tmp, созданной в домашнем каталоге, находятся файлы, изменённые менее недели тому назад.

## 4 **Контрольные вопросы:**

1. Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg...]` Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Строка опций `option-string` – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.
2. При перечислении имён файлов текущего каталога можно использовать следующие символы:
  1. `-` – соответствует произвольной, в том числе и пустой строке;
  2. `?` – соответствует любому одинарному символу;
  3. `[c1-c2]` – соответствует любому символу, лексикографически находящему-



ся между символами `c1` и `c2`.

Например, 1.1 `echo` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; 1.2. `ls.c`–выведет все файлы с последними двумя символами, совпадающими с.с. 1.3. `echoprogram?`–выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.` 1.4.`[a-z]`–соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

3. Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.
4. Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. 16 Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным. Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить

проверять данный блок на других условных выражениях.

5. Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, неравный нулю (т.е. ложь).

Примеры бесконечных циклов: `while true do echo hello andy done` `until false do echo hello mike done`.

6. Строка `if test -f mans/i.s, mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь). При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны.

## 5 Выводы

Таким образом, в ходе ЛРН<sup>№11</sup> я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.